AB 2654/20

TRANSMITTAL OF APPEAL BRIEF (Large Entity)	Docket No. ITL.0038US

In Re Application Of: **John W. Merrill**

JUL 16 2004 (stamp)

Application No. 09/115,359	Filing Date July 14, 1998	Examiner **David D. Knepper**	Customer No. 21906	Group Art Unit 2654	Confirmation No. 1288

Invention: **Automatic Speech Recognition**
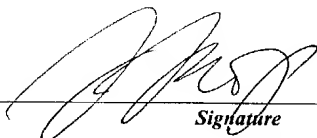
RECEIVED
JUL 1 9 2004
~~Technology Center 2600~~

COMMISSIONER FOR PATENTS:

Transmitted herewith in triplicate is the Appeal Brief in this application, with respect to the Notice of Appeal filed on **May 18, 2004.**

The fee for filing this Appeal Brief is: **NO FEE IS BELIEVED TO BE DUE**

☐   A check in the amount of the fee is enclosed.

☐   The Director has already been authorized to charge fees in this application to a Deposit Account.

☒   The Director is hereby authorized to charge any fees which may be required, or credit any overpayment to Deposit Account No.   **20-1504**

RECEIVED
DEC 1 3 2004
TC 1700

Dated: **July 13, 2004**

_____
Signature

Timothy N. Trop, Reg. No. 28,994
Trop, Pruner & Hu, P.C.
8554 Katy Freeway, Suite 100
Houston, Texas 77024
(713) 468-8880
(713) 468-8883 (fax)

CC:

P30LARGE/REV04

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Applicant:		§	
	John W. Merrill	§	Art Unit: 2654
		§	
Serial No.:	09/115,359	§	
		§	
		§	Examiner: David D. Knepper
Filed:	July 14, 1998	§	
		§	
For:	Automatic Speech Recognition	§	Atty Docket: ITL.0038US
		§	P5634

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

Sir:

Applicant respectfully appeals from the final rejection mailed March 24, 2004.

I. REAL PARTY IN INTEREST

The real party in interest is the assignee Intel Corporation.

II. RELATED APPEALS AND INTERFERENCES

Appeal No. 2001-2630.

III. STATUS OF THE CLAIMS

Claims 33-52 are rejected. Each rejection is appealed.

IV. STATUS OF AMENDMENTS

All amendments were entered.

V. SUMMARY OF THE INVENTION

Referring to Fig. 1, a speech recognition system 11 involves an application software program 10 which needs to respond to spoken commands. For example, the application 10 may be implemented through various graphical user interfaces or windows in association with the Windows® operating system. Those windows may call for user selection of various tasks or control inputs. The application 10 may respond either to spoken commands or tactile inputs. Tactile inputs could include pushing a keyboard key, touching a display screen, or mouse clicking on a visual interface.

The application 10 communicates with a server 12. In an object oriented programming language, the server 12 could be a container. In the illustrated embodiment, the server 12 communicates with the control 14 which could be an object or an ActiveX control, for example. The control 14 also communicates directly with the application 10.

The server 12 can call the speech recognition engine 16. At the same time, a button driver 18 can provide inputs to the server 12 and the control 14. Thus, in some embodiments, the control 14 can receive either spoken or tactile inputs (from the button driver 18) and acts in response to each type of input in essentially the same way.

Referring to Fig. 2, a program for recognizing speech may involve beginning an application (block 90) that needs speech recognition services. The speech engine is provided with a vocabulary of command sets for an active screen or task, as indicated in block 92. The command sets could be the vocabulary for each of the various applications that are implemented by the particular computer system. The command set for the current application that is currently running is communicated to the server 12 or control 14 (block 94). Next, the speech is

2

recognized and appropriate actions are taken, as indicated in block 96. See Specification at page 3, line 10 through page 4, line 29.

Another implementation, shown in Fig. 3, also begins with starting an application, as indicated in block 98. Speech units that need to be decoded are associated with identifiers (block 100). The identifiers may then be associated with a particular action to be taken in the application in response to the spoken command (block 102). Next, the flow determines the identifier for a particular spoken speech unit (block 104). The identifier is provided to a software object such as the control 14, as indicated in block 106. An event is fired when the object receives the command, as shown in block 108. The event may be fired by the object whether the command is a result of a spoken command or a tactilely generated command.

Referring to Fig. 4, the application 10 passes a grammar table to the server 12 (block 20). In particular, the application initializes the grammar with speech identifiers associated with each spoken command used in the application. These commands make up all of the command sets for a given engine. The grammar is a set of commands that may include alternative phrases. For example, a simple grammar could be (start/begin)(navigator). This grammar would respond to the spoken commands "start navigator" and "begin navigator".

The speech recognition engine 16 can operate on phonemes or with discrete terms. Thus, the application provides the particular command set (which is a subset of the engine's available commands) with the active application. This facilitates speech recognition because the speech recognition engine can be advised of the particular words (command set) that are likely to be used in the particular application that is running. Thus, the speech recognition engine only needs to match the spoken words with a smaller sub-vocabulary. For example, if the navigator

function was operating, only the command set of words associated with that application need be decoded. See Specification at page 4, line 30 through page 6, line 4.

In response, the server 12 initializes the speech engine 16 (block 22). The server 12 has a phrase and identifier table 36 as indicated in Fig. 1. The application 10 also sends the speech identifiers associated with given spoken commands to the control 14 or server 12 (block 24). When the control 14 is activated in the container or server, the control may call the OnControlInfoChanged method in the IOleControlSite interface, in an embodiment using ActiveX controls. This provides for transfer of information from the control 14 to the server 12 (block 26). The server in turn may call the GetControlInfo method from the IOleControl interface which allows communications from the server or container 12 to the control 14 (block 28).

The server uses the GetControlInfo method in the IOleControl interface and the OnMnemonic method in IOleControl to request identifiers from the control. The control may provide this information through IOleControlSite interface and the OnControlInfoChanged method, using ActiveX technology for example.

The server 12 enables the speech engine 16 (block 30), for any commands that are active, from the server's table 36. The server uses the table 36 from the application to provide focus in particular applications. The control provides an effect comparable to that of an accelerator key. Namely, it provides a function that can be invoked from any window or frame reference. The application provides the speech identifiers and associates the identifiers with an action by the control. See Specification at page 6, line 5 through page 7, line 2.

4

The server knows which vocabulary to use based on what task is running currently. In a system using windows this would correspond to the active screen. Thus, if the navigator is running, the server knows what the sub-vocabulary is that must be recognized by the speech engine.

When the server receives a speech message, it calls the speech API in the engine 16. When a phrase is detected, the engine provides the phrase to the server for example, as a text message. The container does a table look-up (block 32). On a match between the phrase and the identifier, the server 12 may call the OnMnemonic method of the IOleControl interface, passing the identifier to the control. The control follows its preprogrammed rules and implements the corresponding action (block 34). The control may handle the message internally or send an event to the server.

As a simple example, a given screen may include two buttons, "ok" and "delete". When the application comes up it sends the grammar for this screen to the server. For example, the grammar for "ok" might include "ok", "right" and "correct".

The application then associates "ok" with an identifier which corresponds to a particular control and does the same thing with "delete". The identifier is simply a pointer or handle that is unique, within the application, to the particular command. The table 36 then includes the phrases "ok" and "delete", an identifier for each phrase and an identifier for the control that handles the command.

When a control is instantiated, the application provides it with its identifier. The control is preprogrammed with the action it will take when the server advises the control that its identifier has been called. See Specification at page 7, line 3 through page 8, line 5.

When a speaker uses a word, the speech engine sends the word to the server. The server checks the phases in its table 36 to see if the word is in its active list. In the simple example, if the word sent by the speech engine is not "ok" or "delete," it is discarded. This would indicate a speech engine error. If there is a match between the word and the active vocabulary, the server sends the appropriate control identifier to the appropriate control, which then acts according to its programmed instructions.

A phoneme based speech engine with a large vocabulary can be used with high reliability because the engine is focused on a limited vocabulary at any given time. Advantageously this limited vocabulary may be less than 20 words in the table 36 at any given instance.

This frees the application from having to keep track of the active vocabulary. The server can tell the server which words to watch for at a given instance based on the active task's vocabulary.

There may also be a global vocabulary that is always available regardless of the active screen. For example, there may be a "Jump" command to switch screens or an "Off" command to terminate the active task.

Advantageously, the existing mnemonics or "hot keys" available in Microsoft Windows® may be used to implement speech recognition. For example, the OnMnemonic method may be given the new function of passing information from the server to the control corresponding to a spoken command. See Specification at page 8, line 4 through page 8, line 30.

With embodiments of the present invention, an effect comparable to that of an accelerator key is provided. It gives a focus to the command with reference to a particular application. Therefore, speech can be used to focus between two operating tasks. For example, as shown in

6

Fig. 5, if two windows A and B are open at the same time on the screen 76, the command that is spoken can be recognized as being associated with one of the two active task windows or frames. Referring to Fig. 6, after a command is recognized (block 78), the application provides information about what is the primary, currently operating task and the speech may be associated with that particular task to provide focus (block 80). An input is then provided to one of the tasks (and not the other), as indicated at block 82. In this way, the speech recognition is accomplished in a way which is effectively invisible to the application. To the application, it seems as though the operating system is effectively doing the speech recognition function. Synchronization is largely unnecessary.

The message which is passed to the ActiveX control from the container can include a field which allows the application to know if the command was speech generated. This may be useful, for example, when it is desired to given a spoken response to a spoken command. Otherwise, the application is basically oblivious to whether or not the command was speech generated or tactilely generated. See Specification at page 9, line 6 through page 10, line 1.

While the application loads the identifiers into the ActiveX controls (when they are instantiated), the controls and the container handle all of the speech recognition for the command words. The control and its container are responsible for managing when the words are valid and for sending appropriate messages to the application. Thus, the container or server does all the communication with the speech recognition API. The container may communicate with the ActiveX controls by standard interfaces such as IOleControl. As a result, the number of state errors that would otherwise occur if the application were forced to handle the speech recognition itself. See Specification at page 10, line 2 through page 10, line 13.

VI. ISSUES

A.   Is Claim 33 Anticipated by Trower?

VII.   GROUPING OF THE CLAIMS

All of the claims may be grouped with claim 33.

VIII. ARGUMENT

A.   Is Claim 33 Anticipated by Trower?

In Appeal No. 2001-2630, the Board found that the definition of "object" argued during the appeal was never raised during prosecution. The Applicant respectfully disagrees with that decision. However, to attempt to avoid any further issue in that regard, a continuing application was filed, and amended claims were inserted which incorporated the definition set forth in Applicant's previous Reply Brief. That definition was not considered by the Board for the reasons stated above.

Nonetheless, despite the fact that the definition was included in amended claims, the Examiner simply rejected the claims finally in a first office action on the same grounds.

New claim 33 calls for a method for responding to user inputs to a computer comprising providing a <u>single</u> software object that receives both spoken and non-spoken command information. This claim language leaves no doubt that it is one software object that handles both spoken and non-spoken information. An event is fired when that object receives spoken command information. An event is fired when that object receives non-spoken command information. Thus, as set forth in the claim "the same object can handle both spoken and non-spoken command information."

8

The present claim distinguishes over those systems that use two different apparatus within the system to respond to spoken commands by requiring that a single object handle both spoken and non-spoken commands. An object is defined in Microsoft's Computer Dictionary as "in object oriented programming, a variable comprising both routines and data that is treated as the discrete entity." (Copy attached.)

The analysis of the Board in its decision effectively concurs with the Appellant's argument by suggesting that it is only the system in the Trower reference that responds to both spoken and non-spoken commands. No where in the Board's decision is it ever suggested that an object in Trower does both functions. Specifically, at page 5, the Board explains that "we find that the disclosure of Trower discloses to the artisan that the computer responds to spoken and non-spoken command information as recited in representative claim 14." Of course, this is true and that is the whole point that Appellant was trying to make. The fact that Trower's computer system responds to spoken and non-spoken commands is nothing but the typical prior art. The claim now explicitly requires that a single object within that computer system do both items.

Similarly, the Board's decision on page 5 states that "Trower also discloses that 'server monitors input from the operating system …'." But, again, the fact that the server monitors is not what is now claimed. Finally, the Board concludes that "each of these portions of Trower discloses the artisan that Trower is intended to respond to spoken and non-spoken command information as recited in claim 14." But, of course, this argument is inapplicable to new claim 33, which requires that an object does these functions.

For example, as pointed out on page 4, lines 15-18 of the specification, in some embodiments, the control 14 can receive either spoken or tactile inputs from the button driver 18 and acts in response to each type of input in essentially the same way. In contrast, in the prior art, the information from one type of input goes to one type of control and the information from the other type of input goes to a different type of control.

Therefore, the rejection of claim 33 should be reversed.

IX.   CONCLUSION

Since the rejections of the claims are baseless, they should be reversed.

Respectfully submitted,

Date: <u>July 13, 2004</u>

Timothy N. Trop, Reg. No. 28,994
TROP, PRUNER & HU, P.C.
8554 Katy Fwy, Ste 100
Houston, TX 77024-1805
713/468-8880 [Phone]
713/468-8883 [Facsimile]

APPENDIX OF CLAIMS

The claims on appeal are:

33.   A method for responding to user inputs to a computer comprising:

providing a single software object that receives both spoken and non-spoken command information;

firing an event when said object receives spoken command information; and

firing an event when said object receives non-spoken command information such that the same object can handle both spoken and non-spoken command information.

34.   The method of claim 33 including providing a speech engine with a vocabulary command set for at least two tasks and communicating the appropriate command set for an active task to a speech engine.

35.   The method of claim 33 including associating a command with an identifier and associating the identifier with an action to be taken in response to a command, determining the identifier for a command and providing the identifier to said object.

36.   The method of claim 35 including instantiating said object in a container and communicating the identifier to the object for a command.

37.   The method of claim 36 including communicating information about a first spoken command to the container, checking an active vocabulary list in the container to determine if the first spoken command is one used in an active task, and if the first spoken

command is one used in an active task, transferring the identifier for the spoken command to said object.

38. The method of claim 33 including handling spoken and non-spoken commands in the same way.

39. An article comprising a medium storing instructions that, if executed, enable a processor-based system to:

provide a single software object that receives both spoken and non-spoken command information;

fire an event when said object receives spoken command information; and

fire an event when said object receives non-spoken command information such that the same object can handle both spoken and non-spoken command information.

40. The article of claim 39 further storing instructions that, if executed, enable a processor-based system to provide a speech engine with a vocabulary command set for at least two tasks and communicate the appropriate command set for an active task to a speech engine.

41. The article of claim 39 further storing instructions that, if executed, enable a processor-based system to associate a command with an identifier and associate the identifier with an action to be taken in response to a command, determine the identifier for a command and provide the identifier to said object.

42.   The article of claim 41 further storing instructions that, if executed, enable a processor-based system to instantiate said object in a container and communicate the identifier to the object for a command.

43.   The article of claim 42 further storing instructions that, if executed, enable a processor-based system to communicate information about a first spoken command to the container, check an active vocabulary list in the container to determine if the first spoken command is one used in an active task, and, if the first spoken command is one used in an active task, transfer the identifier for the spoken command to said object.

44.   The article of claim 39 further storing instructions that, if executed, enable spoken and non-spoken commands to be handled in the same way by the same object.

45.   A processor-based system comprising:

a processor; and

a storage coupled to said processor, said storage storing instructions that enable the processor to provide a single software object that receives both spoken and non-spoken command information, fire an event once that object receives spoken command information, and fire an event once that object receives non-spoken command information such that the same object can handle both spoken and non-spoken command information.

46.   The system of claim 45 including an input device, coupled to said processor, to receive spoken commands.

47.   The system of claim 45, said object to handle tactily entered commands and spoken commands in the same way.

48.   The system of claim 45 wherein said storage stores instructions that, if executed, enable the processor-based system to provide a speech engine with a vocabulary command set for at least two tasks and communicate the appropriate command set for an active task to each speech engine.

49.   The article of claim 45 wherein said storage stores instructions that, if executed, enable a processor-based system to associate a command with an identifier and associate the identifier with an action to be taken in response to a command, determine the identifier for a command and provide the identifier to said object.

50.   The article of claim 47 wherein said storage stores instructions that, if executed, enable a processor-based system to instantiate said object in a container and communicate the identifier to the object for a command.

51.   The article of claim 48 wherein said storage stores instructions that, if executed, enable a processor-based system to communicate information about a first spoken command to the container, check an active vocabulary list in the container to determine if the first spoken command is one used in an active task, and, if the first spoken command is one used in an active task, transfer the identifier for the spoken command to said object.

52.   The article of claim 45 wherein said storage stores instructions that, if executed, enable spoken and non-spoken commands to be handled in the same way by the same object.

Microsoft Press
Computer Dictionary

Third Edition

Microsoft Press

object \ob´jekt\ *n.* **1.** Short for object code (machine-readable code). **2.** In object-oriented programming, a variable comprising both routines and data that is treated as a discrete entity. *See also* abstract data type, module (definition 1), object-oriented programming. **3.** In graphics, a distinct entity. For example, a bouncing ball might be an object in a graphics program.

object code \ob´jekt kōd´\ *n.* The code, generated by a compiler or an assembler, that was translated from the source code of a program. The term most commonly refers to machine code that can be directly executed by the system's central processing unit (CPU), but it can also be assembly language source code or a variation of machine code. *See also* central processing unit.

object computer \ob´jekt kəm-pyoo´tər\ *n.* The computer that is the target of a specific communications attempt.

object database \ob´jekt dā´tə-bās\ *n. See* object-oriented database.

Object Database Management Group \ob´jəkt dā´tə-bās man´əj-mənt groop´\ *n.* An organization that promotes standards for object databases and defines interfaces to object databases. *Acronym:* ODMG (O´D-M-G´). *See also* Object Management Group.

object file \ob´jekt fīl´\ *n.* A file containing object code, usually the output of a compiler or an assembler, and the input for a linker. *See also* assembler, compiler (definition 2), linker, object code.

Objective-C \əb-jek´tiv-C´\ *n.* An object-oriented version of the C language developed in 1984 by Brad Cox. It is most widely known for being the standard development language for the NeXT operating system. *See also* object-oriented programming.

object linking and embedding \ob´jekt lēnk´ēng and em-bed´ēng\ *n. See* OLE.

Object Management Architecture \ob´jekt man´əj-mənt är´kə-tek-chur\ *n. See* OMA.

Object Management Group \ob´jekt man´əj-mənt groop´\ *n.* An international organization that endorses open standards for object-oriented applications. It also defines the Object Management Architecture (OMA), a standard object model for distributed environments. The Object Management Group was founded in 1989. *Acronym:* OMG (O´M-G´). *See also* object model (definition 3), OMA, open standard.

object model \ob´jekt mod´əl\ *n.* **1.** The structural foundation for an object-oriented language, such as C++. This foundation includes such principles as abstraction, concurrency, encapsulation, hierarchy, persistence, polymorphism, and typing. *See also* abstract data type, object (definition 2), object-oriented programming, polymorphism. **2.** The structural foundation for an object-oriented design. *See also* object-oriented design. **3.** The structural foundation for an object-oriented application.

object module \ob´jekt moj´ool, mod´yool\ *n.* In programming, the object-code (compiled) version of a source-code file that is usually a collection of routines and is ready to be linked with other object modules. *See also* linker, module (definition 1), object code.

object-oriented \ob´jekt-ōr´ē-en-təd\ *adj.* Of, pertaining to, or being a system or language that supports the use of objects. *See also* object (definition 2).

object-oriented analysis \ob´jekt-ōr-ē-ent-əd ə-nal´ə-sis\ *n.* A procedure that identifies the component objects and system requirements of a system or process that involves computers and describes how they interact to perform specific tasks. The reuse of existing solutions is an objective of this sort of analysis. Object-oriented analysis generally precedes object-oriented design or object-oriented programming when a new object-oriented computer